#IDUGDb2

IDUG Db2 Tech Conference, Rotterdam, Netherlands 2019

JC Greatest Hits, War Stories, and Best Practices 2019 Part 2

John Campbell

IBM Db2 for z/OS Development

Session code: A16

Thursday 24th October, 2019 09:40-10:40

Platform: Db2 for z/OS

IDUG

Leading the Db2 User Community since 1988

Objectives

- Learn recommended best practice
- Learn from the positive and negative experiences from other installations

Agenda

- Insert free space search algorithm
- Insert with APPEND
- Fast Un-clustered insert (Db2 12)
- Hidden ROWID support to partition
- Requirement for increased RID Pool size (Db2 12)
- Increased EDM Pool memory usage for DBDs (Db2 12)
- Setting initial STATISTICS PROFILE (Db2 12)
- System parameter REALSTORAGE_MANAGEMENT
- Transaction level workload balancing for DRDA traffic
- Use of High-Performance DBATs
- Overuse of UTS PBG tablespaces and MAXPARTS
- IRLM query requests for package break-in

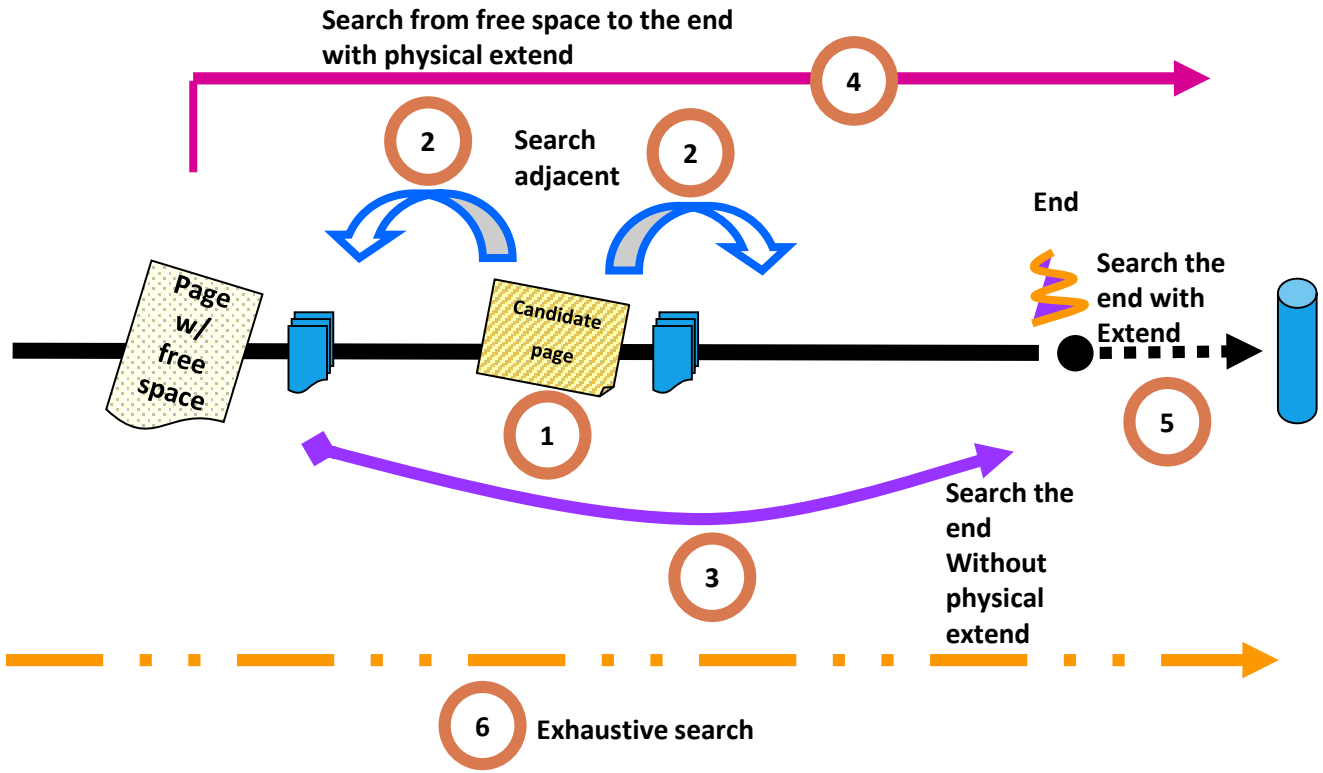
Insert free space search

- **Insert performance is a trade off across**
  - Minimising CPU resource consumption
  - Maximising throughput
  - Maintaining data row clustering
  - Reusing space from deleted rows and minimising space growth
- **Warning: insert space search algorithm is subject to change and it does**

Insert free space search …

- **For UTS and classic partitioned table space:**
  1. Index Manager will identify the candidate page (next lowest key rid) based on the clustering index
     - If page is full or locked, skip to Step 2
  2. Search adjacent pages (+/-) within the segment containing the candidate page
     - For classic partitioned it is +/-16 pages
  3. Search the end of pageset/partition without extend
  4. Search the space map page that contains lowest segment that has free space to the last space map page up to 50 space map pages, skip to Step 5
     - This is called "smart exhaustive space search"
  5. Search the end of pageset/partition with extend until PRIQTY or SECQTY reached
  6. Perform "exhaustive space search" from front to back of pageset/partition when PRIQTY or SECQTY reached
     - Very expensive i.e., space map with lowest segment with free space all the way through
- **For classic segmented table space, steps are very similar except the sequence:**
  - 1->2->3->5->4->6

Insert – free space search steps (UTS and classic partitioned tablespace)



INSERT with APPEND

- **With APPEND**
  - Always insert into the end (e.g., at end of last part of of UTS PBG)
  - No attempt to use free space created by deleting rows
  - Space search will always start on last data page of the last space map page
  - Search (minus) n pages prior to the last data page of the last space map page
  - n pages determined by SEGSIZE (use large SEGSIZE)
- **Ideal for application journal and event logging tables with little or no update or delete**
- **APPEND can be beneficial as it avoids the "normal" space search**
- **With high concurrency may not see any improvement due to space map contention at the end of table**
  - So in data sharing environment with high concurrency, APPEND should be combined with MEMBER CLUSTER for tables which require faster insert and do not need to maintain the data row clustering
    - Similar to V8 MEMBER CLUSTER, PCTFREE 0, FREEPAGE 0 for classic partitioned tablespace
- **With LOCKSIZE PAGE|ANY, space growth can be excessive with partially filled data pages**
  - With LOCKSIZE ROW, the space growth should be slower
- **When using LOAD with DUMMY, do NOT use PREFORMAT option**
  - DASD space size will be reduced after LOAD
- **When using REORG (with DISCARD), do NOT use PREFORMAT option**

Fast Un-clustered INSERT

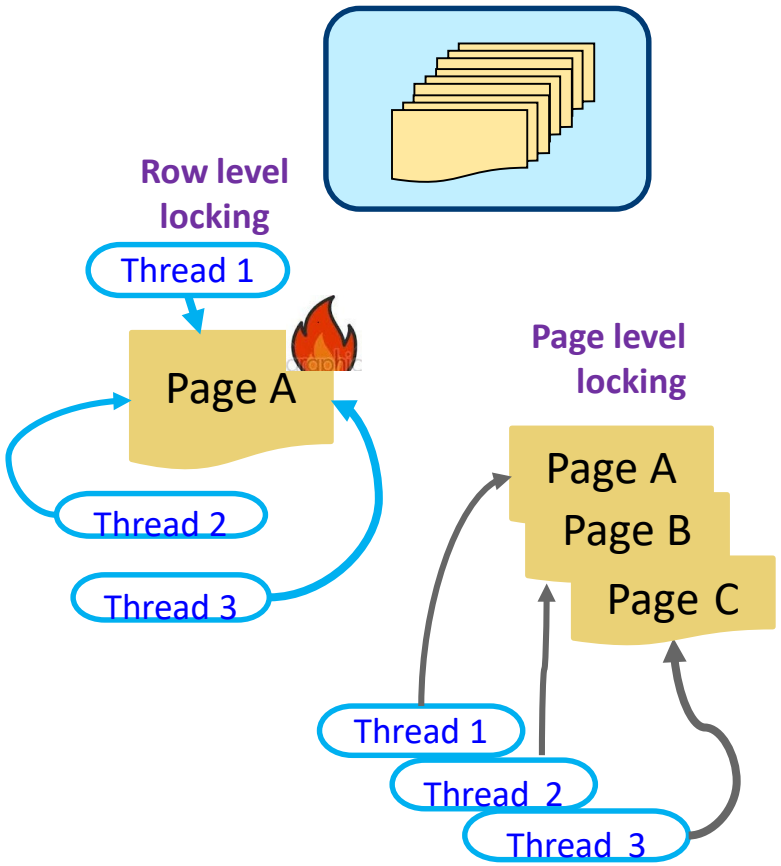
- **Insert workloads are amongst the most prevalent and performance critical**
- **Performance bottleneck will vary across different insert workloads**
  - Index maintenance?
  - Log write I/O?
  - Data space search (space map and page contention, false leads)
  - Format write during dataset extend
  - PPRC disk mirroring
  - Network latency
  - etc.
- **Common that index maintenance and/or log write I/O time may dominate and mask any insert speed bottleneck on table space**

Fast Un-clustered INSERT …

- **Officially referred to as "Insert Algorithm 2 (IAG2)"**
- **Potentially delivers significant improvement for un-clustered inserts (e.g., journal table pattern) where**
  - Heavy concurrent insert activity (many concurrent threads)
  - Space search and false leads on data is the constraint on overall insert throughput
- **Applies to any UTS table space defined with MEMBER CLUSTER**
  - Applies to both tables defined as APPEND YES or NO
- **Implemented advanced new insert algorithm to streamline space search and space utilisation**
  - Eliminates page contention and false leads
    - Space is preallocated/preassigned in order to fill up pipes which are pulled from
    - Set of pipes allocated per partition per Db2 member
    - Space allocation occurs at first pageset open and real time when there is space shortage in each individual pipe
  - Default is to use the new fast insert algorithm for qualifying table spaces
    - DEFAULT_INSERT_ALGORITHM system parameter can change the default
    - INSERT ALGORITHM table space attribute can override system parameter

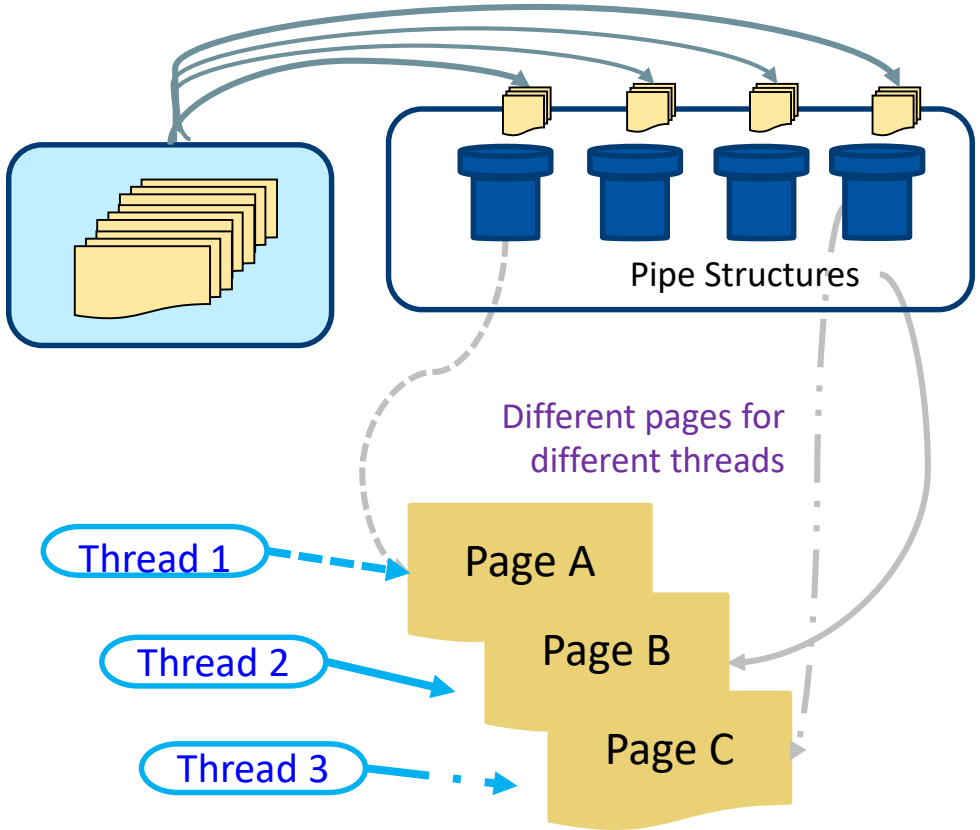
Fast Un-clustered INSERT …

Insert Algorithm 1 concepts



Insert Algorithm 2 concepts

Base on space information, pages are pre-distributed into different pipes



Fast Un-clustered INSERT …

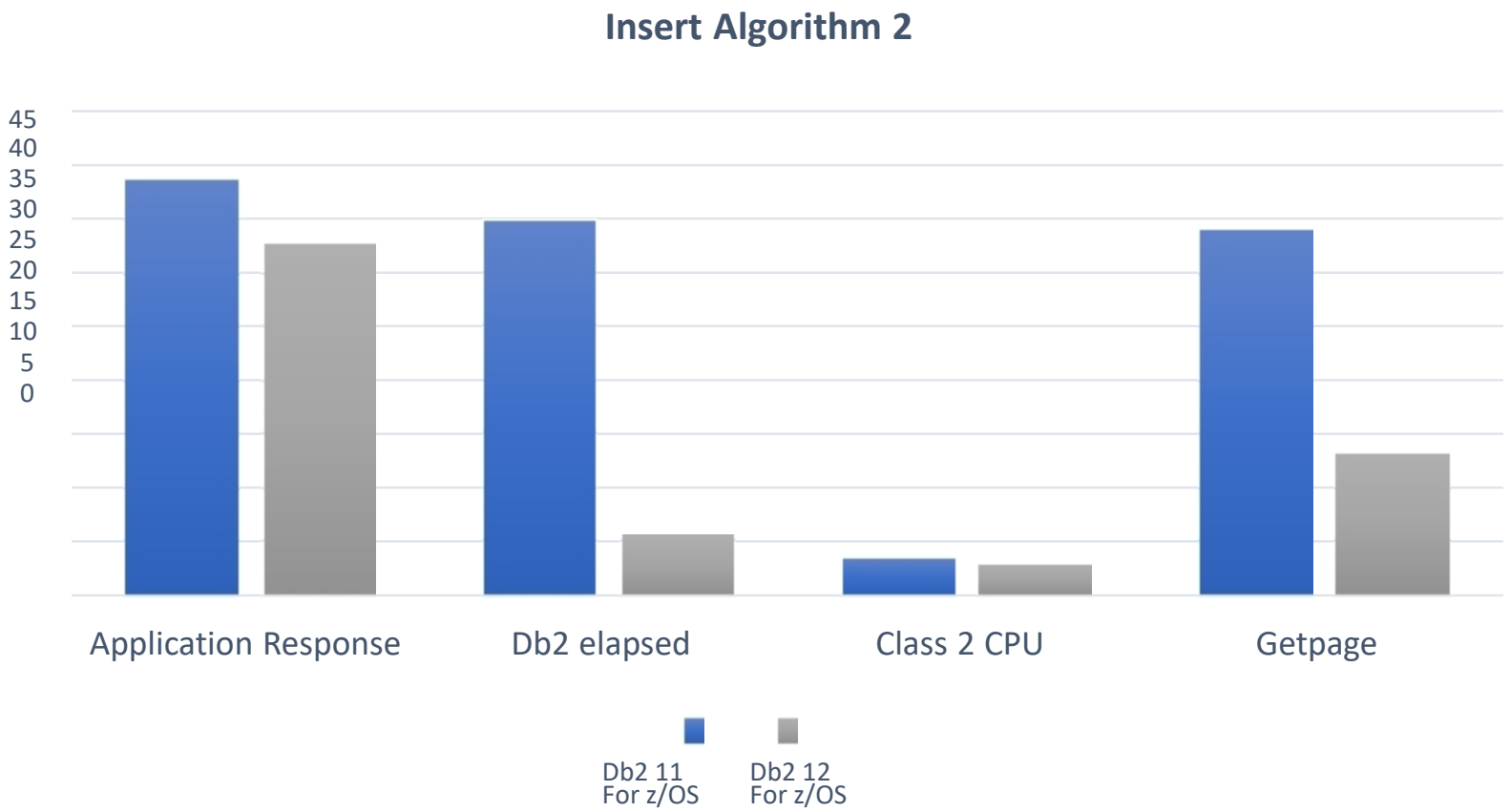
- **Your mileage will vary**
  - Many insert workloads will see no improvement and is to be expected
  - Will probably not see much difference/improvement when only one insert per commit scope
  - Some specific insert workloads may see significant improvement
  - Less benefit as more indexes are added to the respective table
- **Will shift the bottleneck to the next constraining factor**
- **LOAD SHRLEVEL CHANGE can also use Fast Un-clustered INSERT**
- **Fast Un-clustered INSERT will be disabled when lock escalation occurs or use of SQL LOCK TABLE**
- **First available after new function activation (FL=V12R1M500)**
  - Open to change in behavior for insert processing at new function activation

Fast Un-clustered INSERT …

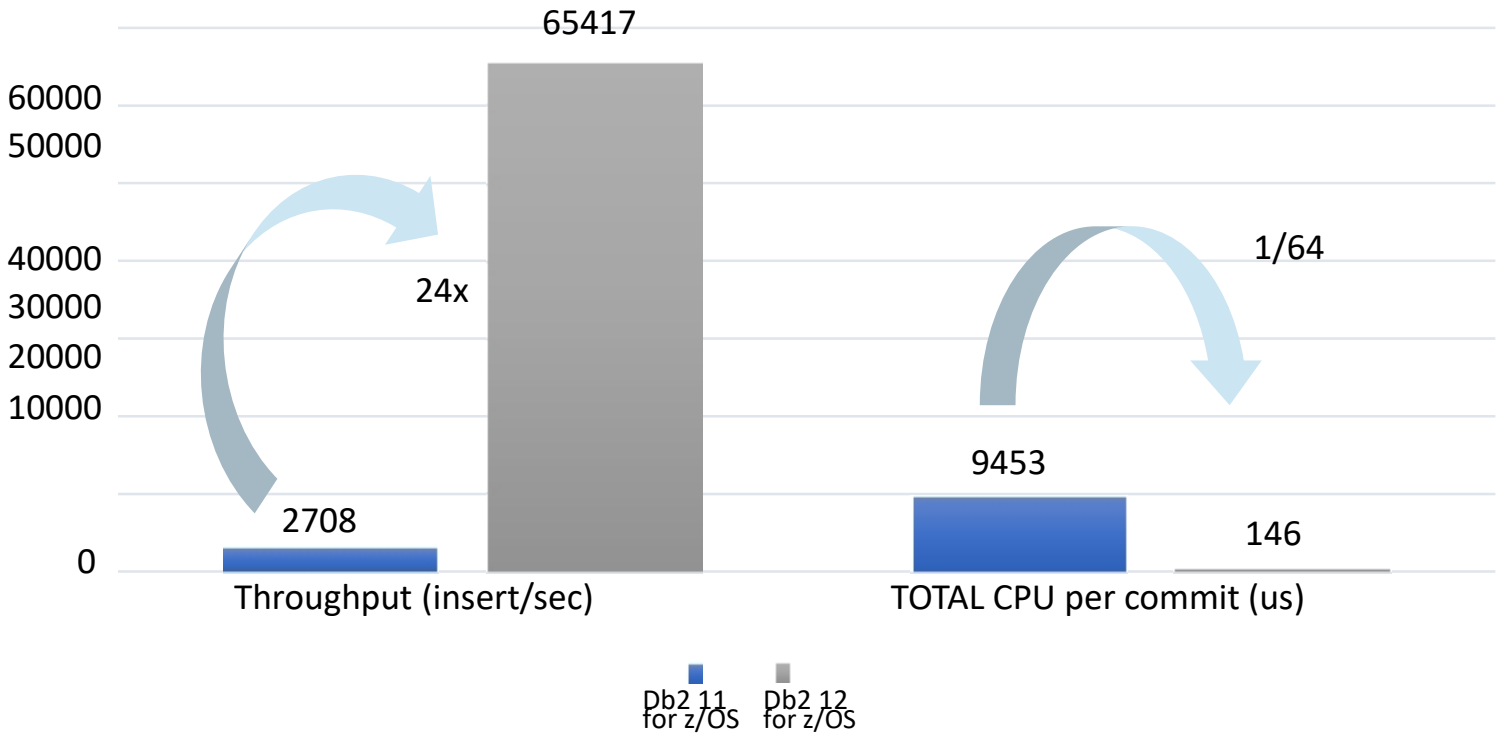
- **Must be very aggressive on applying preventative service tagged with "IAG2" for robustness and serviceability**
- **Strongly recommend the apply of PTF for APAR PH02052 (Closed) which implements automatic re-enablement with retry logic**

- **Current JC point-in-time recommendation**
  - One size probably does not fit all tablespaces
  - Change system wide default - set system parameter DEFAULT_INSERT_ALGORITHM = 1 (old basic insert algorithm)
  - Use INSERT ALGORITHM 2 (new fast insert algorithm) selectively at individual table space level to override system wide default
  - Additional benefit may be gained where effective CICS and DRDA thread reuse coupled with running packages bound with RELEASE(DEALLOCATE)

Fast un-clustered INSERT – Shifting The Bottleneck …



Fast un-clustered INSERT – Shifting The Bottleneck …



UTS PBG with MEMBER CLUSTER, RLL, with 400 bytes per row, one index,

800 concurrent threads, 10 insert per commit

Hidden ROWID support to partition

- **Make a sequential insert a random insert to reduce "hot spots"**
- **ROWID (generated random number) can be used as a partitioning column**
- **Application impact if ROWID cannot be hidden**
  - APARs to support hidden ROWID
    - PI76972, PI77310, PI77302 (Db2 12)
    - PI77718, PI77719, PI77360 (Db2 11)
- **Benefits**
  - Allows table to be partitioned where no natural partitioning key exists or candidate partitioning keys do not provide a good spread across partitions
  - Transparent to the application
  - Improved insert throughput
    - Less lock/latch contention on index and data

```
CREATE TABLE  PRDA.ZJSCNTP0
      (  CLIENT    VARGRAPHIC(3) NOT NULL,
         WI_ID     VARGRAPHIC(12) NOT NULL,
         LENGTH    SMALLINT,
         DATA      VARCHAR(1000),
         ROW_ID    ROWID NOT NULL
         IMPLICITLY  HIDDEN generated always
      )  PARTITION BY (ROW_ID)
          (PARTITION 1 ENDING AT (X'0FFF'),
           PARTITION 2 ENDING AT (X'1FFF'),
           PARTITION 3 ENDING AT (X'2FFF'),
           PARTITION 4 ENDING AT (X'3FFF'),
           :
           PARTITION 14 ENDING AT (X'DFFF'),
           PARTITION 15 ENDING AT (X'EFFF'),
           PARTITION 16 ENDING AT (MAXVALUE))
```

Requirement for increased RID Pool size (Db2 12)

- **Increased space requirement for RID Pool as a result of RID value increase 5 -> 8-byte value**
  - Internally Db2 for z/OS uses a normalised 8-byte RID value to allow for future expansion
  - More RID blocks will be used for the same query because each RIDLIST holds fewer RIDs
  - RID Pool memory usage will be roughly 60% higher (for smaller lists it will be up to 2x higher)
  - Should plan on increasing MAXRBLK (RID Pool size) by up to 60% before leaving Db2 11
  - Data Manager logical limit (RIDMAP/RIDLIST) reduced from 26M RIDs to 16M RIDs
  - More RID Pool overflow to workfile is to be expected
- **Increased space requirement for RID Pool as a result of potentially more use of list prefetch**
  - Enhancement to the Optimizer cost model to more closely reflect the true cost (and benefit) of list prefetch
  - Expected to see an increase in list prefetch (and potentially hybrid join)
  - But not necessarily changes in the access plan where Db2 would previously have chosen a sort avoidance plan
  - Db2 for z/OS trying to be careful not to select list prefetch (with sort) as an access path when there was an alternative access path that could use an index to avoid a sort i.e., for pagination type SQL

Increased EDM Pool memory usage for DBDs (Db2 12)

- **In Db2 12 there will be increased demand for EDM pool memory for DBDs**
  - How much depends on the size and number of OBDRECs (tables) and the number of OBDPSETs (table spaces)
- **Extra memory comes from "puffing" of table obdrec and tablespace obdpset to new DBD format in Db2 12**
  - Pre-Db2 12 "puffed" size under Db2 12 > Db2 12 size (after ALTER/REPAIR) > Db2 11 size
- **The "puffed" OBD (OBDREC and OBDPSET) will not be written out to DBD01 which means that "puffing" to Db2 12 format will occur every time the DBD is read from DBD01**
- **Before APAR PH05624, the "puffing" code is invoked even when the OBD is already in Db2 12 format**
  - No further additional memory requirement since OBD (OBDREC and OBDPSET) is already in Db2 12 format
- **Some customers experienced significant performance impact after migration to Db2 12**
- **Solutions**
  - Double size of EDM DBDC pool before leaving Db2 11
  - Apply PTF for APAR PH05624
  - Any DDL or REPAIR DBD REBUILD will result in update of OBD to Db2 12 format and new format will be persisted in DBD01

Setting initial STATISTICS PROFILE (Db2 12)

- **Statistics profile is automatically created on first BIND/REBIND/PREPARE/EXPLAIN after entry to Db2 12 for z/OS**
- **Statistics profile created based on all the existing statistics in the Catalog**
  - Stale or inconsistent “specialised” statistics may exist in the Catalog that have not been collected for a long time
  - Statistics no longer collected because either too expensive to collect with RUNSTATS or were ineffective in solving access path problems, and not ever cleaned out
- **“Specialised” statistics will now be MERGED into the respective statistics profile and will then be collected again on every execution of RUNSTATS with USE PROFILE**
- **After the initial profile create, cannot tell from the subject statistics profile what statistics are the ones that were the older/inconsistent statistics**
- **Stale or inconsistent statistics may already be causing sub-optimal access path choices prior to Db2 12**

Setting initial STATISTICS PROFILE …

- Use the following sample query to identify the inconsistent statistics

```
SELECT TYPE, NUMCOLUMNS, TBOWNER, TBNAME, NAME, MIN(STATSTIME),COUNT(*)
FROM SYSIBM.SYSCOLDIST CD
WHERE STATSTIME < CURRENT TIMESTAMP - 1 MONTH
AND (TYPE IN ('C', 'H') OR NUMCOLUMNS > 1
  OR STATSTIME < CURRENT TIMESTAMP - 1 YEAR)
AND NOT EXISTS
(SELECT 1
FROM SYSIBM.SYSINDEXES I
WHERE I.TBCREATOR = CD.TBOWNER
AND   I.TBNAME = CD.TBNAME
AND   CD.STATSTIME BETWEEN I.STATSTIME - 8 DAYS
                       AND I.STATSTIME + 8 DAYS)
AND NOT EXISTS
(SELECT 1
FROM SYSIBM.SYSTABLES T
WHERE T.CREATOR = CD.TBOWNER
AND   T.NAME = CD.TBNAME
AND   CD.STATSTIME BETWEEN T.STATSTIME - 8 DAYS
                       AND T.STATSTIME + 8 DAYS)
GROUP BY TYPE, NUMCOLUMNS, TBOWNER, TBNAME, NAME
ORDER BY TYPE, NUMCOLUMNS, TBOWNER, TBNAME, NAME
WITH UR;
```

Setting initial STATISTICS PROFILE (Db2 12) …

- **So it is important to clean up any (SYSCOLDIST) statistics that you do not intend to regularly collect before first BIND/REBIND, PREPARE or EXPLAIN after entry to Db2 12 for z/OS**
  - Simplest way to find these is to look for tables with rows having different STATSTIME values in SYSCOLDIST
  - Reset access path statistics back to default using RUNSTATS with RESET ACCESPATH option
    - Sets relevant Catalog column values to -1, and clears out entries in SYSCOLDIST
  - Run “regular” RUNSTATS after the RESET operation

System parameter REALSTORAGE_MANAGEMENT

- **The frequency of contraction mode is controlled by system parameter REALSTORAGE_MANAGEMENT**
- **Db2 uses DISCARD with KEEPREAL(YES) = "Soft Discard"**
- **Unless you have ample real memory headroom and can tolerate some memory growth, recommend running with REALSTORAGE_MANAGEMENT=AUTO (default)**
  - With RSM=AUTO, Db2 will regularly discard unused REAL memory frames
    - RSM=AUTO with no paging (AUTO-OFF) → "Soft Discard" at Thread Deallocation or after 120 commits
    - RSM=AUTO with paging (AUTO-ON) → "Soft Discard" at Thread Deallocation or after 30 commits – STACK also DISCARDED
  - Pros of the "Soft Discard"
    - Reduced REAL memory use
    - **Reduced exposure to SPIN lock contention**
  - Cons:
    - CPU overhead in MSTR/DIST SRB time (which can be greatly minimised with good thread reuse)
      - Worse on IBM z13 and z14 hardware
    - **64-bit shared and common real memory counters are not accurate until paging occurs**
      - **The memory is only "virtually freed"**
      - **RSM flags the page as freed or unused, but the frame is still charged against Db2**

System parameter REALSTORAGE_MANAGEMENT …

- **REALSTORAGE_MANAGEMENT = OFF**
  - Do not enter 'contraction mode' unless the system parameter REALSTORAGE_MAX boundary is approaching OR z/OS has notified Db2 that there is a critical aux shortage
- **But … provided an installation is certain about the generous allocation of REAL memory and avoiding any paging at all times then setting to OFF can generate CPU savings where there is poor thread reuse and/or surge of DBAT termination**
  - CPU savings can be seen in Db2 MSTR and DIST system address spaces
- **Setting system REALSTORAGE_MANAGEMENT = OFF requires a long-term commitment to maintaining generous REAL memory "white space" at all times**

Transaction level workload balancing for DRDA traffic

- **Re-balancing mechanism to adjust the workload distribution across Db2 members to meet the goal**
- **Workload distribution can be non-uniform, can by spiky, can see "sloshing" effect**
- **Drivers up to today's latest available level have the two main issues**
  - Balancing tends to favor highest weighted member(s)
    - Less load: Driver starts traversing through server list and always selects the first Db2 member in the server list, because it is available to take a new transaction based on a non-zero weight
    - Heavy Load: Driver starts traversing through server list from first Db2 member and checks whether the Db2 member is available to take a new transaction based on the weight
  - Failover to another Db2 member has caused client driver to excessively retry
- **Client driver team have made the following changes to address issues:**
  - Driver knows what is the last Db2 member used, so it starts from next Db2 member
    - For first iteration through server list, driver will use every Db2 member once
    - Second iteration onwards the driver choses the Db2 member based on the weight and number of transactions already ran
  - Failover to another Db2 member will only be attempted once and the DataSource address will be used

Transaction level workload balancing for DRDA traffic …

- **Recommendations (prioritised) to improve the workload distribution across both members**
  1. Validate whether or not the goal of DDF transactions are being met under normal operating conditions and adjust the WLM policy accordingly (adjust the goal so it can be met and/or break down the DDF transactions into multiple service classes)
  2. Upgrade to new Db2 Connect driver level V11.1 M4 FP4 (JDBC 4.0 spec) which will provide a new workload balancing distribution algorithm which is much better – new workload balancing algorithm only available for JDBC 4.0
  3. Only if necessary consider setting RTPIFACTOR in IEAOPTxx (e.g. 50%) to reduce the possibility of a sudden redirection of workload to the alternate Db2 member when PI > 1
- **Supporting recommendations**
  - With sysplex workload balancing (sysplexWLB) enabled, CONDBAT on each member should be set higher than the sum of all possible connections across all connection pools from all application servers plus 20% to avoid reduction in health of Db2 member
    - 80% of CONDBAT=Health/2, 90% of CONDBAT=Health/4
  - MAXDBAT should be set to a value which permits normal workload levels, AND allow for peaks, AND possible Db2 member outage … but not so high as to allow a ‘tsunami’ of work into the system
  - MAXCONQN can be used as the “vehicle” to limit queuing → force early redirection of connections wanting to do work to the another Db2 member

Use of High-Performance DBATs

- **High-Performance DBATs have the potential to provide a significant opportunity for CPU reduction by**
  - Avoiding connection going inactive and switching back to active later
  - Avoid DBAT being pooled at transaction end i.e., going back into DDF thread pool for reuse by probably a different connection
  - Supporting true RELEASE(DEALLOCATE) execution for static SQL packages to avoid repeated package and statement block allocation/deallocation
- **Very powerful performance option, but can be dangerous … if not used intelligently**

Use of High-Performance DBATs …

- **Important operational considerations**
  - Do not use RELEASE(DEALLOCATE) on common widely shared packages across distributed workloads as it will considerably drive up the requirement for MAXDBAT
    - Risk of not having enough DBATs available for DRDA workloads that are not using High-Performance DBATs
    - Worst case running out of DBATs completely or escalating thread management costs
  - Check that all the ODBC/JDBC driver packages in the existing collection NULLID are bound as RELEASE(COMMIT) – otherwise they must be rebound with RELEASE(COMMIT)
    - WARNING: Since V9.7 FP3a, default BIND option for Db2 client driver packages has been RELEASE(DEALLOCATE)!
  - Have another collection (e.g., NULLID2) where the Db2 client driver packages are bound with RELEASE(DEALLOCATE)
    - Applications wanted to use high performance DBATs should pint to this collection ID from the data source (JDBC) or database configuration file (ODBC)
  - Be very careful about DDF workloads re-using common static SQL packages used by CICS and/or batch workloads bound with RELEASE(DEALLOCATE)
  - Do not over-inflate the application server connection pool definitions, otherwise it will considerably drive up the demand for High-Performance DBATs

Use of High-Performance DBATs …

- **Recommendations**
  - Consider rebinding with RELEASE(DEALLOCATE) the static SQL packages that are heavily-used and unique to specific high-volume DDF transactions
  - Enable High-Performance DBATs by using the -MODIFY DDF PKGREL(BNDOPT|BNDPOOL)
  - Be prepared to use -MODIFY DDF PKGREL(COMMIT)
    - Allow BIND, REBIND, DDL, online REORG materialising a pending ALTER to break in
    - Switch off High-Performance DBATs at first signs of DBAT congestion i.e. overuse of DBATs

Overuse of UTS PBG tablespace and MAXPARTS

- **Primary driver for the developing UTS PBG tablespace was the removal of the 64GB limit for classic segmented tablespace and avoid the disruptive migration to classic partitioned tablespace**
- **Some considerations**
  - All indexes are going to be NPIs
  - Limited partition independence for utilities (REORG, LOAD)
  - No partition parallelism for utilities (LOAD, UNLOAD, REORG)
  - Partitioning not used for query parallelism
  - Degraded insert performance (free space search) as the number of partitions grow
  - If REORG a partition list/range, it may encounter undetected deadlock between applications and REORG during the SWITCH phase (i.e. drain and claim in different order)
  - REORG PART will fail for a full UTS PBG partition if FREEPAGE or PCTFREE are non-zero
  - Setting system parameter REORG_DROP_PBG_PARTS = ENABLE could lead to operational issues if the number of PARTs are pruned back
    - No point-in-time recovery prior to the REORG that prunes partitions
    - Cannot use DSN1COPY to move data between Db2 systems
- **Should not be using UTS PBG as the design default for all tables (with large number of partitions)**

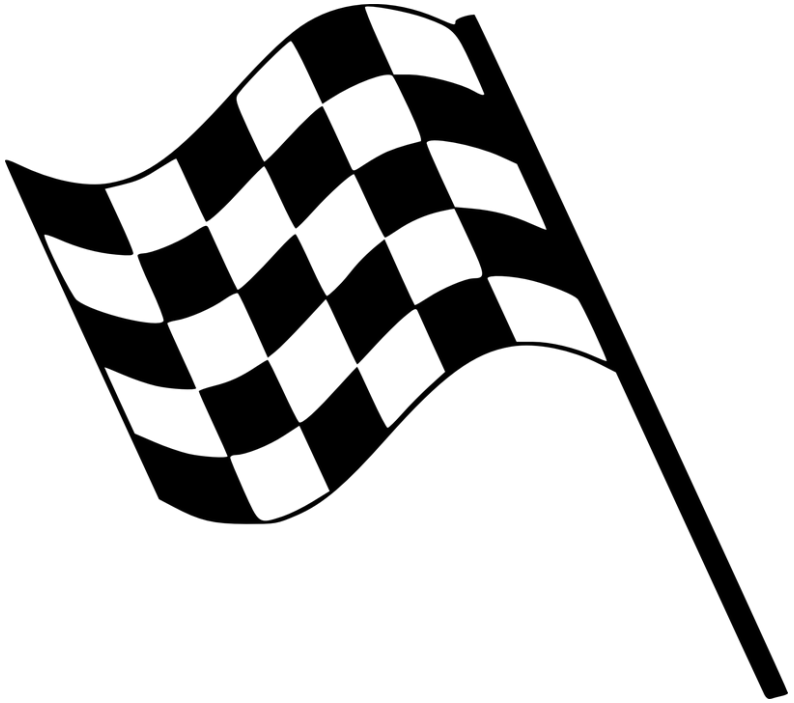
Overuse of UTS PBG tablespace and MAXPARTS …

- **General recommendations for use of UTS PBG tablespace**
  - Only use UTS PBG tablespace as the alternative and replacement for classic segmented tablespace
  - A table greater than 60GB in size should be created as a UTS PBR tablespace
  - Good reasons to limit number of partitions - should have as few partitions as possible - ideally only 1
  - DSSIZE and SEGSIZE should be consistent with the target size of the object e.g.
    - Small size object: DSSIZE = 2GB and SEGSIZE = 4
    - Medium size object: DSSIZE = 4GB and SEGSIZE = 32
    - Large size object: DSSIZE = 64GB and SEGSIZE = 64
  - REORG at the table space level unless do not have sufficient DASD space for sort
  - Set system parameter REORG_DROP_PBG_PARTS = DISABLE
    - If required to prune back the number of partitions
      - Use online system parameter to temporarily enable for controlled use
    - Better still, in Db2 12, use the DROP_PART YES option of REORG

IRLM query requests for package break-in

- An IRLM query request is used to see if there are X-package lock waiters on a package during commit as part of commit phase 1 broadcast
- The break-in at commit is charged to Acctg Class 2 CPU Time for both local allied and DBATs regardless under read-only or update transactions
- The IRLM query request is “fast path” and a single query request has little or no performance impact
- There is an IRLM query request per RELEASE(DEALLOCATE) package per commit (per lock token basis)
- If there are many RELEASE(DEALLOCATE) packages loaded by commit time in a long-running thread, then there will be an IRLM query request for each one of those packages at that time
- It is possible to accumulate many distinct RELEASE(DEALLOCATE) packages across 1000 successive commits on each CICS-Db2 thread i.e., REUSELIMIT is 1000 in CICS definitions, and many different transactions running against the same DB2ENTRY in CICS definitions, and multiple packages per commit
- A big multiplier exaggerates the CPU overhead of IRLM query requests
- Likely to be a problem when applications are using many fine-grained packages for IO routines e.g., CA Gen (CoolGen)
- Solution: set system parameter PKGREL_COMMIT=NO and toggle on/off when needed

Questions?



Please fill out your session evaluation before leaving!

John Campbell

IBM Db2 for z/OS Development

campbelj@uk.ibm.com

Session code: A16

IDUG

Leading the Db2 User Community since 1988